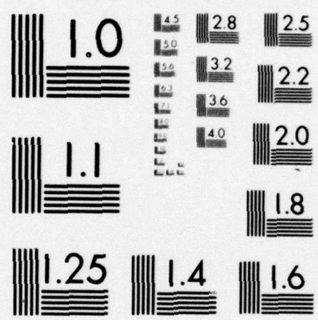
AD-A069 809 NAVAL SURFACE WEAPONS CENTER DAHLGREN LAB VA F/G 9/2
INPUTP (GENERAL PURPOSE INPUT PROCESSOR) USER GUIDE.(U)
JAN 79 P J GOYETTE
UNCLASSIFIED NSWC/DL-TR-3880 NL

1 OF 1
AD
A069809

END
DATE
FILMED
7 -79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NSWC TR-3880	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 INPUTP (GENERAL PURPOSE INPUT PROCESSOR) USER GUIDE.		5. TYPE OF REPORT & PERIOD COVERED 9 Final rept.,
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 Peter J./Goyette		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Surface Weapons Center Code K71 Dahlgren, Virginia 22448		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS N0003079WR92417 9K50TR001
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Surface Weapons Center (K71) Dahlgren, Virginia 22448		12. REPORT DATE 11 January 1979
		13. NUMBER OF PAGES 48
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 14 NSWC/DL-TR-3880		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited

12 49p.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Input processing, Input processor, Default data, Data bases, Free-form, Table-driven processing, Data-driven processing, Mnemonic input, Scripting, Input overriding

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The General Purpose Input Processor (INPUTP) program is a very flexible, entirely data-driven utility that may be used as an input preprocessor to any FORTRAN program. The INPUTP program reads two external files to define the names and default values for all user input. Then through input, the user may override the default values for any input parameters.

next page

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

LB

391 598

The input processor generates two FORTRAN block data routines containing DATA statements to define the input parameters. These routines must be compiled and loaded to initialize all input variables.

All data is read by the input processor in a "free-form" format, which simplifies the creation and maintenance of the data files and greatly reduces the chances for user input errors.

The INPUTP program is coded in the SIMSCRIPT II.5 language, and is operational on the Control Data Corporation (CDC) 6700 computer system under the SCOPE 3.4 operating system.

FOREWORD

For large computational models whose requirements are frequently changing, it is highly desirable to simplify the mechanics of processing model input. The General Purpose Input Processor (INPUTP) program provides a means of eliminating the ever-present "INPUT" routines by acting as a stand-alone, data-driven input preprocessor. It provides the model users with a convenient, flexible input mechanism, and a formatted report of the input environment.

Most importantly, however, it provides a standard mechanism for performing program input, and eliminates the redundant and incompatible input procedures heretofore associated with computational models.

The INPUTP Computer Program was prepared in the Ballistic Sciences Branch of the Computer Programming Division. The programming effort expended on this project was funded by the FBM Geoballistics Division under project number: N0003079WR92417.

This report was reviewed by Ira V. West, Head of the Ballistic Sciences Branch of the Computer Programming Division.

Released by:

Ralph A. Nieman

RALPH A. NIEMANN, Head
Strategic Systems Department

ABSTRACT

The General Purpose Input Processor (INPUTP) Program is a very flexible, entirely data-driven utility that may be used as an input preprocessor to any FORTRAN program. The INPUTP Program reads two external files to define the names and default values for all user input. Then, through input, the user may override the default values for any input parameters.

The Input Processor generates two FORTRAN block data routines containing DATA statements to define the input parameters. These routines must be compiled and loaded to initialize all input variables.

All data is read by the input processor in a "free-form" format, which simplifies the creation and maintenance of the data files and greatly reduces the chances for user input errors.

The INPUTP Program is coded in the SIMSCRIPT II.5 language, and is operational on the Control Data Corporation (CDC) 6700 computer system under the SCOPE 3.4 operating system.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

SECTION 1. GENERAL DESCRIPTION

The Input Processor Program (INPUTP) is a general purpose input processor which may be used as a stand-alone preprocessor program to any FORTRAN computational program. It initializes all input variables to their default value, processes user override input, and prints an Initial Conditions Report showing the value of all input variables prior to beginning execution, thereby eliminating the need for lengthy "INPUT" routines within computational programs. The resultant savings in core required for execution of the computational program could be significant. Not only may programmers take advantage of this input processor for models still in the planning stage, but existing models could be adapted to use INPUTP with a minimum of effort.

The INPUTP Program itself requires no programming modifications to be used by any model as an input processor. It is entirely data-driven by two external files which may be created by the model programmers or users. These files declare the legal input variables and define the default values for all input. This feature greatly simplifies program maintenance on the part of the model programmers. Even before the computational model has been updated to handle a new input parameter, the variable mnemonic and default value can be added to the appropriate files, and the input variable will be available as soon as the model is ready for it. The model programmers do not have to spend time modifying FORMAT statements in input and output routines, and user deck set-ups are not affected by the introduction of a new input parameter.

The INPUTP Program generates two FORTRAN block data routines which contain DATA statements to define all input parameters. When these routines are loaded, all input parameters will be defined to their desired value.

Taking advantage of the fact that it is a separate program, users may execute INPUTP alone to check the validity of their input, prior to actually executing the computational program. This would allow users the chance to review their input prior to beginning a long, and expensive, computer run. For models with a moderate-sized input data base, this test run could be executed via a remote terminal.

1.1 Purpose of This User Guide

The objective of this user guide is to provide the information necessary for users to interface their computational program with the Input Processor and to prepare input to the program.

1.2 System Configuration

Simply speaking, the Input Processor initializes all input variables to their default value, processes user input to override any of these default values, and then generates two FORTRAN block data routines to data-define all input parameters. Thus, by compiling and loading these block data routines

prior to loading and executing the associated computational program, all input parameters will be initialized to their desired value. The input processor also generates an updated, or "dynamic", default data base which reflects the value of all input variables after user overrides have been processed. In this way, the current environment is preserved for multi-case runs.

The INPUTP Program and its associated computational program must be loaded and executed in sequence for each case in the user's job set-up. With the appropriate job control procedures, as explained in Section 4, this approach is hidden from the users.

The Input Processor Program is completely compatible with the CDL (Configuration Description Language) Processor Program, which may also be used as a preprocessor to a computational program. (See Reference 1 for more information on the CDL Processor).

1.3 Program Environment

The Input Processor Program is coded in the CDC SIMSCRIPT II.5 language, Version 4.0. The coding conforms to the programming standards outlined in Reference 2.

1.4 Equipment Environment

The program is currently executing at NSWC on the CDC 6700 and Cyber 74 computers under control of the SCOPE 3.4 operating system.

The features of the CDC SCOPE 3.4 BEGIN/REVERT mechanism (originally conceived and implemented by the University of Washington, and modified locally) are also used for flexible interaction between INPUTP and the user computational program.

1.5 Project References

The philosophy and design of the Input Processor evolved during the design and implementation phases of two large simulation programs - TRICS (Trident Computational Simulation) and AWSS (Advanced Weapon System Simulation). The implementation of these two projects required a uniform input mechanism that was also highly user-oriented, and one that could respond to frequent modifications without requiring coding changes. The Input Processor Program evolved from these concepts.

SECTION 2. INPUT DESCRIPTION

2.1 General Description

The design of the Input Processor Program, which was generally described in Section 1.2, enables it to be used by most FORTRAN programs as an input utility without requiring any special programming considerations or naming conventions within the computational program.

The information needed by the Input Processor to effectively process model input and perform the appropriate consistency checking (valid mnemonics, table sizes not exceeded, etc.) is contained on a data file that is defined by the programmers and read by the Input Processor. This "Initialization File" defines the class names (see Sections 2.2.1 and 2.2.2) that are applicable to the particular model, and defines the table names and gives their maximum dimension sizes. Section 4.4 lists the error conditions that are detected by the Input Processor.

Also required is a default data base that contains a default value for all variables and tables mentioned on the Initialization File. This Default File may also contain additional variables or tables that were not mentioned on the Initialization File; however, they will be ignored by the Input Processor. In this way, the users may create a "worst case" Default File that can be used for several models.

This approach enables the same Input Processor to process any model input simply by reading a unique Initialization File for each model. The result is an Input Processor that is extremely flexible and usable. It eliminates the need for each model to have its own unique "INPUT" routine, or to re-code an Input Processor. Instead, a user can just create or modify a data file.

As a further indication of its flexibility, the information on the Default File and the Initialization File is read under a "free-form" format, which minimizes errors in defining these data bases.

2.2 Organization and Description of Input Data Bases.

As previously mentioned, the Input Processor Program is entirely driven by the contents of the Initialization File and Default File. Both files are easily created and can be quickly modified as changes are made to the computational program. Initially, both files would probably be created by the model programmers, since they would be most familiar with input mnemonics, table sizes, FORTRAN common block names, etc. However, the users of the model can easily create their own Default Files to reflect various scenarios.

The following two sections describe the specific contents and format requirements of the Initialization and Default Files. Generally speaking, all information may be provided in a "free-form" format, with only keywords beginning in column 1.

2.2.1 Initialization File (SIMU11)

The Initialization File (or "Template") is read by the Input Processor to define the environment of the model. As mentioned, this file is initially created by the programmers of the associated computational model and generally should only be modified by them, since the information on this file is directly related to the code in the model. The accessing of this file would normally be hidden within the BEGIN/REVERT procedure for executing the model, and would be invisible to the user.

This file normally resides on the model UPDATE Program Library (OLDPL) file, which is a random access file. If so, the Initialization File must be retrieved via the CDC UPDATE utility (see Reference 3) and input to the Input Processor as a sequential BCD file. (If retrieving this file from an OLDPL, use "UPDATE,D,8" to suppress UPDATE identifiers on the "COMPILE" File.) However, the Initialization File may reside on any BCD file.

The information on this file is not format-dependent, with the exception that column 1 is reserved for keywords, which are described below. It is also required that there be at least one blank column between fields. The keywords are not order-dependent, either, and may be repeated on the same Initialization File. This allows several Initialization Files, possibly from different programs or different OLDPL'S, to be merged into one large file before being read by the Input Processor.

Table I below lists the keywords that may be provided on the Initialization File and gives the data requirements associated with each keyword. A detailed explanation of each line is given below it in Table II.

A sample Initialization File is shown in Appendix A.

Table 1. Keywords and Data Requirements (Initialization File)

LINE NO.	KEYWORD AND ASSOCIATED PARAMETERS				
1	CLASS NAME1				
2	MNEMONIC1	MNEMONIC2	MNEMONIC3		
3	VECTORS				
4	MNEMONIC1	MNEMONIC2	MNEMONIC3		
5	TABLES				
6	MNEMONIC1	N.DIMS	N.ROWS	N.COLS	N.BLOCKS
7	MNEMONIC2	N.DIMS	N.ROWS	N.COLS	N.BLOCKS

Table 1. Keywords and Data Requirements (Initialization File) (Continued)

LINE NO.	KEYWORD AND ASSOCIATED PARAMETERS
8	PACKED.TABLES
9	MNEMONIC1 PFACT N.DIMS N.ROWS N.COLS N.BLOCKS
10	MNEMONIC2 PFACT N.DIMS N.ROWS N.COLS N.BLOCKS
11	PRINT.OPTIONS
12	MNEMONICS NAME1 VALUE1
13	REPORT.TYPE NAME2 VALUE2
14	BLOCK.DATA
15	STARTS
16	SUBROUTINE HEADER1
17	/CBNAME1
18	COMMON/CBNAME1/MNEMONIC1, MNEMONIC2, ETC.
19	ENDS
20	STARTT
21	SUBROUTINE HEADER2
22	/CBNAME2
23	COMMON/CBNAME2/ MNEMONIC1, MNEMONIC2, ETC.
24	ENDT
25	TITLE.ARRAY
26	NAME
27	OCTAL.TABLES
28	NAME1 NAME2 NAME3
29	OCTAL.SIMPLES
30	NAME1 NAME2 NAME3
31	HEX.TABLES
32	NAME1 NAME2 NAME3
33	HEX.SIMPLES
34	NAME1 NAME2 NAME3

Table II. Detailed Explanations of Keywords and Data Requirements

LINE NO.	EXPLANATION
1	Each class name that has been defined to categorize simple variable input must be mentioned. Class names may be from 1-20 characters, and duplicate class names may appear on the file. (See Section 2.2.2 for a further discussion of class names.)
2	The mnemonics for all simple variables within the above mentioned class name are listed. The mnemonics may be from 1-7 characters. Duplicate mnemonics within the same class name are ignored.
3	All vector input (i.e., 3-element arrays) are declared under the class name VECTORS.
4	The mnemonics (1-7 characters) for all input vectors must be given following the class name VECTORS. Duplicate mnemonics within this class name are ignored.
5	The class name TABLES is required to define the mnemonics and maximum dimension sizes for all tables.
6-7	The mnemonic (1-7 characters) for each table is given followed by the number of dimensions in the table (N.DIMS), and each dimension size - number of rows (N.ROWS), number of columns (N.COLS), number of 2-D blocks (N.BLOCKS). Duplicate table names will be ignored if their dimension sizes agree with the sizes previously declared. If not, an error message will be printed.
8	The class name PACKED.TABLES may be used to declare tables containing several values per computer word.
9-10	The same information is given as is required for other tables (see lines 6-7). However, a packing factor (PFACT) is also required. PFACT is the number of values that will be stored per word. The dimension sizes should be given as if the table were not packed.
11	The keyword PRINT.OPTIONS allows the user to override the default conditions concerning the printing of the Initial Conditions Report (see Section 3.3).

LINE NO.	EXPLANATION
12	Within the PRINT.OPTIONS section, a key name called MNEMONICS is recognized. If, after all user input is processed, the input variable NAME1 equals VALUE1 (an alpha value), the input mnemonics will not be printed in the Initial Conditions Report. If multiple MNEMONICS sections are present, the last name and value read are retained. (Section 3.3 contains a more detailed description of this option.)
13	The name REPORT.TYPE is also recognized within the PRINT.OPTIONS section. If the input variable NAME2 equals the value VALUE2, a full Initial Conditions Report will be printed for every case. If multiple REPORT.TYPE sections are read, the last name and value are retained. (Section 3.3 provides a fuller description of this option.)
14	The BLOCK.DATA keyword denotes the start of information needed to generate the appropriate FORTRAN block data routines. (Note: The block data routines that are generated will be either true block data subprograms or subroutines containing DATA statements, depending on whether the subroutine header card (lines 16 and 21) is a BLOCK DATA card or a SUBROUTINE card.)
15	The STARTS keyword denotes the start of the common blocks that will be needed in the block data routine for simple variables and vectors.
16	The line following STARTS must be the subroutine header card that will be used in the block data routine for variables and vectors. Only the first header card is saved when multiple STARTS sections are encountered.
17	Each common block must be preceded by a line containing a slash (/) in column 1, immediately followed by the common block name. Duplicate common blocks within the same start-end section are bypassed.
18	A FORTRAN common block that will be needed in the block data subroutine. All information up to the next common block name (as in line 17) will be copied verbatim into the block data routine.
19	The ENDS keyword marks the end of the common blocks for simple variables. Multiple STARTS-ENDS sections may occur within a BLOCK.DATA section.

LINE NO.	EXPLANATION
20	The STARTT keyword denotes the start of the common blocks that will be needed in the block data routine for tables.
21	Following the STARTT keyword must be the subroutine header card that will be used in the block data routine for tables. Only the first header card is saved when multiple STARTT sections are encountered.
22	(same as line 17)
23	(same as line 18)
24	The ENDT keyword marks the end of the common blocks for tables. Multiple STARTT-ENDT sections may occur within a BLOCK.DATA section.
25	The keyword TITLE.ARRAY is used to declare the name of an array into which the Input Processor will data-define the user's title.
26	The FORTRAN array name which will contain the case title. This array should be declared as an 8 word integer array in the computational program, and the common block that contains the array should be provided within the STARTT-ENDT section. However, the array should not be declared under the keyword TABLES.
27	The keyword OCTAL.TABLES is used to declare those tables whose values will be input as octal numbers.
28	The mnemonic of each table whose input values will be read under an octal format is given. These tables must have previously been defined under the TABLES class (see lines 5-7).
29	The OCTAL.SIMPLES keyword is used to define each simple variable whose value will be input as an octal number.
30	The mnemonic of each variable whose value will be input as an octal number must be mentioned. These variables must have already been declared under some class name (see lines 1-2).
31	The keyword HEX.TABLES is used to declare those tables whose values will be input as hexadecimal numbers.

LINE NO.	EXPLANATION
32	(Same as line 28, only for hexadecimal tables)
33	The HEX.SIMPLES keyword is used to define each simple variable whose value will be input as a hexadecimal number.
34	(Same as line 30, only for hexadecimal variables)

It is recommended that this Initialization File be stored as a DECK on the model OLDPL file to assure that the common blocks needed are up-to-date. This is assured if the model programmers structure their OLDPL with common blocks as UPDATE COMDECKS. Then, with *CALL'S on the Initialization File, a quick update can be performed to write the initialization deck to the COMPILE File. (This methodology is described in Reference 2.)

If a model does not have any simple variable input, or any table input, the appropriate START-END section can be omitted. In this case, the corresponding block data routine will not be generated.

2.2.2 Default File (SIMU13)

The Default File contains the default data values for all input variables and tables. Normally, this file will be stored as a sequential BCD file on the CDC 6700 permanent file system. However, the users may maintain various versions of a default data base corresponding to different scenarios.

The data on the Default File is segregated according to unique "class names". Each class name is intended to be a meaningful descriptor of the variables within that class. It is hoped that this approach will make the input, and output, more visible to the user and more relevant to the physical sense of the problem.

However, it was decided beforehand that all input tables would be included in the class name TABLES, and any input vectors would be included in the class name VECTORS. The remaining input, the simple variables, may be broken up into many class names.

If the CDL Processor Program is also being used as a preprocessor, one must avoid using class names that are keywords to the CDL Processor. Reference 1 defines the keywords recognized by the CDL Processor.

It is the Initialization File that controls what is read from the Default File. As explained in Section 2.2.1, the Initialization File must contain the mnemonics for all input variables and tables, as well as their class names. If a class name is read from the Default File that did not appear on the Initialization File, all variables within that class name are skipped over. Similarly, when reading mnemonics and values within a class

name, if a mnemonic is read that was not declared under that class name on the Initialization File, it is also skipped over. While this approach could allow an error to go undetected, it is felt that the benefits in being able to use one large default data base for many models, each with its own unique Initialization File, outweighed that disadvantage. However, there is an execution option (described in Section 4.2) that will cause an informative message to be printed when class names or mnemonics are being bypassed. On the other hand, if a simple variable or vector that was declared on the Initialization File is not found on the Default File, an error message will be printed. (This test is not performed for tables, for reasons that will be explained in Section 3.2.2).

Normally, if a variable or table is defined more than once on the Default File, the last value overrides the previous. However, there is another execution option that will cause duplicate mnemonics to be ignored. (This option is also explained in Section 4.2).

The following tables show the composition of a Default File. The data requirements for each keyword are shown in Table III, and an explanation for each line is given below it in Table IV.

Again, the data on this file does not have to follow any particular format, with the exception that only keywords (i.e., class names) begin in column 1 and all data be separated by at least one blank column. Also, the class name for table data (TABLES) must be the last keyword on the file.

A sample Default File is shown in Appendix B.

Table III. Keywords and Data Requirements (Default File)

LINE NO.	KEYWORD AND ASSOCIATED PARAMETERS			
1	CLASS NAME			
2	MNEMONIC	VALUE	DEFINITION	
3	VECTORS			
4	MNEMONIC		DEFINITION	
5	VALUE(1)	VALUE(2)	VALUE(3)	
6	TABLES			
7	MNEMONIC	N.ROWS	DEFINITION	
8	VALUE(1,1,1)	VALUE(1,2,1)	VALUE(1,3,1)	ETC.
9	VALUE(2,1,1)	VALUE(2,2,1)	VALUE(2,3,1)	ETC.

Table IV. Detailed Explanations of Keywords and Data Requirements

LINE NO.	EXPLANATION
1	The class name to describe the simple variables which follow. Class names may be from 1-20 characters.
2	Each variable associated with the above class name is defined by giving its mnemonic and value. The remainder of the card is available for an optional definition. A mnemonic may be from 1-7 characters (although 7 character names are non-ANSI). Its value may be real, integer, alpha, octal or hexadecimal.
3	VECTORS is the class name to define real input vectors.
4	The mnemonic of a vector, followed by an optional definition.
5	The real values of the three components of the above mentioned vector.
6	The class name to define input tables.
7	The table mnemonic followed by the actual number of rows (1st dimension size) to be input, and optionally followed by a definition of the table.
8-9	The table values are input "row-wise", i.e., all columns are input for row 1, then all columns for row 2, etc. The example shows how a 3-dimensional table would be input. (Additional description on table input is given in Section 2.3.4)

2.3 Organization and Description of User Input

Through the normal input file, the user may change any default input values to construct the proper environment for each case. However, prior to processing any user input, the Input Processor initializes all input variables and tables by reading the Default File. Thus, the user should only have to override a small subset of the total user input to define the initial parameters for his run.

The user input may consist of one or many cases, each of which will be a separate execution of the computational program. Within each case, the keywords that the user provides determine how the data following is processed. Except for three special keywords - OLDDEFAULT/NEWDEFAULT and END - the keywords may appear in any order and they may be repeated within a case.

A "case" refers to a particular set of input parameters for one execution of the computational program. If the user desires to execute the computational program for several different values of an input parameter, he must define a new case for each new value.

Unless the user directs the Input Processor to return to the original Default File (via the OLDDEFAULT keyword), each succeeding case in the job set-up will define the initial default values as those that were in effect from the previous case. Then it will process the user overrides for this case. Thus, the initial conditions for each case are a summation of the original default values and all user overrides for all previous cases (with the latest overrides taking precedence).

The following sections describe all of the keywords that are recognized by the Input Processor, and describe the data requirements associated with each one. All keywords, and only keywords, begin in column 1. However, data associated with the keywords can appear anywhere else on the card, and as much data as desired can be put on one card (provided there is at least 1 blank column between data items).

A note of caution: when providing data in scientific notation (E-format) there should not be any blanks between the number and exponent fields. For example, input "2.85E-4" and not "2.85 E-4".

It may be helpful to refer to the sample input illustrated in Appendix C to clarify the description of each keyword.

2.3.1 OLDDEFAULT/NEWDEFAULT

The OLDDEFAULT keyword can be used to direct the input processor to return to the original Default File to initialize input parameters for the case being run. The NEWDEFAULT keyword denotes that the cumulative, or "Dynamic", Default File should be used; however, this is the normal condition whether or not NEWDEFAULT is used.

There is no data associated with this keyword.

This keyword, if used, must be the first card of the input record for the case.

2.3.2 TITLE

The TITLE keyword allows the user to input a descriptive title to document the case being run. The 80 column card immediately following this keyword will be read as the user title. The title will appear as a page header on each page of the Initial Conditions Report for the case.

The title will be passed to the computational program (via the block data routines) if a TITLE.ARRAY keyword was declared on the Initialization File (see Section 2.2.1).

The title will not be carried over to succeeding cases.

2.3.3 VECTORS

The keyword VECTORS is used to input vector quantities. The mnemonics associated with this keyword are defined when the Initialization File is read.

As with all keywords, VECTORS begins in card column 1. It is followed by the mnemonic and three components of the vector that the user wishes to override. These parameters may appear in any card column (except column 1) separated by at least one blank column.

2.3.4 TABLES

All tables are input under the keyword TABLES. The dimensionality and maximum size of each table has been defined via the Initialization File. Thus the user just has to give the table mnemonic and the actual table size he will be providing (i.e., the number of rows), which may be less than the maximum size defined. For a single dimensioned array, the "number of rows" is just its length. However, for multi-dimensioned arrays, for each row that is read, the Input Processor expects the full number of columns to be supplied. As on the Default File, tables are read row-wise.

For example, assume a table dimensioned as ABC(I,J,K). Then for each row that the user inputs (up to the maximum of "I" rows), he must supply all "J" elements (columns). Three-dimensional tables are considered to be blocks of 2-dimensional tables, and the user may input as many blocks as he wishes, up to the maximum of "K".

2.3.5 Class Names

All simple variables are input under their respective class name. The mnemonics associated with each class name are defined via the Initialization File (see Section 2.2.1). The class name, as with all keywords, begins in column 1. On the cards following it are mnemonic-value pairs for as many variables in this class name as the user wishes to override. The user may provide as many pairs per card as he desires, as long as each data item is separated by at least 1 blank column. Input variables of all modes (real, integer, or alpha) may be mixed freely within a class.

2.3.6 END

The keyword END denotes the end of data for the case. Any data following the END card will be applied to the next case.

This keyword is optional on the last case, or if only running one case.

SECTION 3. OUTPUT DESCRIPTION

3.1 General Description

The Input Processor Program produces two different types of user output. Its major function is to generate output files that contain source code suitable for input to the FORTRAN compiler. These files, of course, contain the block data routines that will data-define the input parameters for the computational program. As briefly mentioned in Section 1.2, the INPUTP Program also generates an updated default data file which will be used to define the default values on the next case (if the NEWDEFAULT option is in effect). This Dynamic Default File could easily be read by non-FORTRAN programs which might not be able to utilize the block data concept.

The other type of output produced by the Input Processor is an Initial Conditions Report. This is a printed report showing the value of all input parameters prior to beginning execution of the computational program. This report also flags those variables and tables that were modified by user overrides.

3.2 Organization and Description of User Output Files

The files described in this section are of concern only to the programmers or analysts responsible for interfacing the Input Processor Program with the computational program. Normally, the manipulation of these files would be handled within a BEGIN/REVERT procedure for program execution, and would be hidden from the users of the computational program.

Naturally, the exact contents of each of the following files depends entirely on the contents of the model Default File and Initialization File. Also, the block data files will only be opened if they are needed.

If an error occurs during the processing of user input, the block data routines and the Dynamic Default File are not generated.

A sample of generated block data routines may be found in Appendix E.

3.2.1 Block Data Routine For Variables and Vectors (SIMU21)

The block data routine to define simple variables and vectors is written to the file SIMU21. The subroutine header for the routine is copied exactly as it appeared on the Initialization File. Each unique common block that appeared within a STARTS-ENDS section on the Initialization File is also copied to this file. Following them, the Input Processor generates FORTRAN DATA statements to data-define the input variables and vectors.

Each class name is written as a comment prior to defining the variables within that class. Real variables are defined using an "E(21.14)" format, integers are defined with an "I14" format, and alpha variables are defined using a "10H" notation. Octal and hexadecimal variables are defined by writing their 20-character octal value, suffixed by a "B".

For vectors, each unsubscripted array name is written followed by its three real values.

An "END" statement terminates the block data routine.

On the first case, or any case where the user specified the OLDDEFAULT keyword, the block data routine for variables is generated. For other cases, however, this routine is only generated if the user actually provided override input for a variable or vector.

The INPUTP Program does not perform any integrity checking to insure that each simple variable being defined is included in one of the common blocks copied to the block data routine. It is the user's responsibility to check the FORTRAN cross reference map for "Stray Names" after compiling the block data routines.

3.2.2 Block Data Routine For Tables (SIMU23)

The block data routine to define input tables is written to the file SIMU23. It contains the subroutine header card and common blocks that were provided on the Initialization File. The formats under which it generates the DATA statements are the same as those mentioned above for simple variables. However, packed tables are defined as 20-character octal words, suffixed by a "B".

For 1-dimensional arrays, each element of the array is defined with a DATA statement. For 2-dimensional arrays, an implied "DO" loop is used to define each row of the array. Three-dimensional arrays are defined similar to 2-dimensional arrays, with a third subscript added to identify the block number.

If any tables are defined on the Default File at less than their maximum size (i.e., less than the maximum dimension sizes specified on the Initialization File), the remaining elements will not be defined. The Input Processor only generates DATA statements to define the greatest number of elements as were actually input via the Default File or user overrides. However, if the user inputs a table with fewer elements than were defined on the Default File, the remaining elements will retain their original value.

Like the simple variables, a block data routine for tables is always generated on the first case, or any case in which the user specified OLDDEFAULT. It will also always be generated if a TITLE.ARRAY keyword was specified on the Initialization File. However, the contents of this routine will vary according to which execution option was chosen.

As will be explained in Section 4, the INPUTP Program may be executed with the parameter TABLES=YES or TABLES=NO. (The default condition is TABLES=NO.) This option controls whether or not table data are read from the Default File. As one might guess, TABLES=YES implies that the user wants the default tables read from the Default File, whereas TABLES=NO will cause the Input Processor to stop reading from the Default File when the keyword TABLES is encountered. (Hence, the reason for requiring that TABLES be the last keyword on the Default File.)

Anytime the block data routine for tables is generated it will contain DATA statements to define all input tables.

However, under the default condition (TABLES=NO), only those tables that were changed via user overrides will be written to the block data routine. If no tables are modified on the first case, a block data routine is generated, but it only contains the common blocks for tables; no DATA statements are generated. This approach assumes that the user has previously generated and saved a block data routine for tables which will initially be loaded to set the default values for all tables. Then at execution time, any tables that are modified and written to the table block data routine can also be loaded to override the default values. For computational models having a large number of tables, this approach is much more economical. An example of the job control to implement this option is given in Section 4. If this approach is used, however, the user must insure that both block data routines do not have the same name, or they will not both be loaded. A possible way to control this is to use UPDATE "IF DEF" directives (see Reference 3) around the subroutine header cards on the Initialization File.

3.2.3 Dynamic Default File (SIMU17)

The Dynamic Default File is primarily a local file used by the Input Processor Program to save the latest version of the default data base from case to case. As mentioned, however, this file could be accessed by the computational program if needed for some special application.

The format of this file is similar to that of the normal Default File (SIMU13) except that the comment field is not present. To save space, several mnemonic-value pairs are written per line.

If the Input Processor is executed with the TABLES=NO option, the Dynamic Default File will only contain those tables that were changed by user overrides.

3.3 Initial Conditions Report

The Initial Conditions Report shows the status of all input parameters prior to beginning the computational program for each case. On the first case, a full report is printed. A full report lists the values of all simple variables in each class name, the values of all vectors, and (if executed with TABLES=YES) the values of all tables. (In the default mode, only those tables that are modified by the user are listed.) The class names for simple variables are listed in the order in which they were encountered on the Initialization File, while the mnemonics within each class are listed in alphabetical order. The simple variables are followed by vectors, and then by tables, with each listed alphabetically. As an added feature, the report also indicates the variables or tables that were changed by user overrides. Those variables are denoted by an asterisk (*) immediately before the mnemonic.

One-dimensional tables are printed across the page, four values per line. Two-dimensional tables are listed by row, with the row number printed in the left margin. For three-dimensional tables, the block number is given followed by its associated 2-dimensional table.

By default, on all cases after the first, only those variables or tables that were modified by the user will be printed. If the user wishes to override this default condition, he may declare an input variable on the Initialization File to control this feature (see line 13 in Table 1, Section 2.2.1). If a variable is declared under this REPORT.TYPE option,the INPUTP Program will test its value prior to printing the Initial Conditions Report. If its current value equals the value specified on the Initialization File, a full Initial Conditions Report will be printed.

Similarly, the user may also declare an input variable to control the printing of the mnemonics of the input parameters (see line 12 in Table 1, Section 2.2.1). If such a variable is specified and its current value equals the value given on the Initialization File, the printing of mnemonics will be suppressed; only the value of the input parameters will be printed.

Appendix D contains sample output showing a full Initial Conditions Report such as one might receive on the first case, and a reduced report that one would receive on subsequent cases.

SECTION 4. EXECUTING THE PROGRAM

4.1 General Description

As mentioned already, a convenient way of executing INPUTP and the associated computational program is to create a B⁻ IN/REVERT procedure for program execution. In this way, the execution of the INPUTP Program and the compilation and loading of the block data routines it generates can be made automatic. Without BEGIN/REVERT, the control cards needed by the user could be cumbersome, and job control errors could easily occur.

Also, it is necessary to load and execute INPUTP and the computational program for each case in the user's job set-up. This requires looping through a series of control cards for an unknown number of cases. However, this also is very easily handled by a BEGIN/REVERT procedure.

The following section explains the control cards needed to execute INPUTP and the computational program.

4.2 Sample Job Control and User Options

The control cards needed for utilizing the INPUTP Program as an input preprocessor are illustrated in Appendix F as they might appear in a BEGIN/REVERT procedure. Actually, three procedures are shown; the first procedure (MODELXEQ) would be "called" by the user, and it would invoke the other two. The INIT procedure attaches any program files or data files that are needed by INPUTP or the computational program (hereafter referred to as "MODEL") for the execution of all cases. The EXEC procedure executes INPUTP and "MODEL" for each case. It also contains the necessary logic to decide which, if any, block data routines need to be compiled. The EXEC procedure is a recursive procedure; it begins itself for each case.

Each line in the BEGIN/REVERT procedures listed in Appendix F is identified by a line number. By referencing these numbers, the options available to the user will be described. A familiarity with BEGIN/REVERT syntax is assumed.

Although the job control within the BEGIN/REVERT procedures may seem excessively complicated for executing a simple computational program, the control cards which the user supplies could not be any simpler. They are -

```
ATTACH,PROFIL,BEGIN/REVERT PROCEDURE FOR EXECUTION.
BEGIN,MODELXEQ.
```

These two cards are all that are necessary for a user to execute as many cases of "MODEL" as he supplies.

As shown on Line 1 (referring to Appendix F), the top level procedure contains several parameters which the user may override. These parameters indicate the local file names for user input and "MODEL" output, and the

INPUTP options associated with reading the Default File. These parameters are passed to the EXEC procedure, which actually executes INPUTP and "MODEL". Any other parameters which might be needed to facilitate model execution could easily be added to the MODELXEQ call line.

The initialization procedure (lines 8 - 20) is only executed once. It is invoked (line 2) to attach all program files and data files which may be needed during the execution of the Input Processor or the computational model. Also, all local files that will be used within the BEGIN/REVERT procedures are returned to prevent possible conflicts. Line 10 shows the attach of the Input Processor Program itself, INPUTP.

The procedure for model execution (lines 21 - 45) is invoked by MODELXEQ on line 3. This procedure executes the Input Processor, compiles any block data routines that were generated, loads the relocatable binary files resulting from the compilations, and loads and executes the computational model. It then tests (line 39) for the presence of the dummy file SIMU2. If this local file does not exist, the EXEC procedure begins itself to execute another case (line 40). When SIMU2 does exist, the "IF" test will fail and the procedure will revert to its caller (line 42), which will eventually be the MODELXEQ procedure. As one may have guessed, the local file SIMU2 is created by the Input Processor when it detects the last case in the user's job set-up. Thus, the multiple executions of INPUTP and MODEL are hidden from the user and automatically controlled.

In connection with the generated block data routines, the local files SIMU21 and SIMU23 must be returned before each case (line 22), and the files that will contain the relocatable binary code of these routines must be rewound (line 23).

On the execution of the Input Processor (line 23), the local files used for input (SIMU5) and output (SIMU6) are made known to INPUTP. Also, the value of the "PARM" parameters are passed to INPUTP. (The usage and consequences of the "PARM" parameter were explained in Sections 2.2.2, 3.2.2 and 3.3.) The three parameters are summarized in the following table, and their default value denoted. The user may specify as many of the parameters as needed, or none at all.

Table V. "PARM" Parameters and Default Values

PARAMETER	VALUE	EXPLANATION
LASTV	YES	Retain the last value when duplicate mnemonics are present on the Default File. (default)
	NO	Keep the first value read for all variables on the Default File.
TABLES	NO	Stop reading the Default File when the keyword TABLES is encountered. (default)
	YES	Read the entire Default File.

PARAMETER	VALUE	EXPLANATION
WARN	NO	Do not print any warning messages. (Default)
	YES	Print a warning message when unknown mnemonics are encountered on the Default File.

On the first case, SIMU21 and SIMU23 will always be generated, so both "IF" tests (lines 26 and 30) will be true and both block data routines will be compiled. The files LGO21 and LGO23 will contain the binary code resulting from these compilations (lines 28 and 32). They are then loaded (lines 35 and 36) to initialize all input parameters.

On succeeding cases, if a new block data routine is generated, it will be compiled and its binary code will replace that from the previous compilation. If no new compilation is necessary, the files LGO21 and LGO23 still contain the results from the last compilation and can simply be re-loaded.

If the user executes with the default of TABLES=NO, he must have previously saved the relocatable binary file for the table block data routine. This file would be attached in the INIT procedure (line 16) and loaded (line 34) prior to loading the routine containing override tables.

The XEQ parameter in the BEGIN/REVERT procedure illustrates the input validation feature mentioned in Section 1. When the value of XEQ is NO, lines 26 - 37 will be bypassed and the computational program will not be executed.

4.3 Execution Statistics

Execution time and core requirements for INPUTP are entirely dependent on the size of the model data base. The number of variable names and table names that are declared on the Initialization File determines the amount of time that will be spent searching and accessing lists of names. Also, the I/O time for reading a large default data file will affect overall execution time.

A data base containing 100 simple variables and 300 table elements would take about 5-10 seconds to process, while a data base of 300 simples and 3000-4000 table elements could take about 50 seconds.

The minimum core requirements for INPUTP are about 50K octal words of memory. Memory requirements will increase as INPUTP processes the model data. The actual core requirements for any particular model will depend on both the number of input variables and, more importantly, the size of the input tables.

4.4 Error Messages and User Response

The Input Processor Program checks the validity of the data supplied by the user as well as the data on the Default File. Whenever an error is detected, INPUTP prints a message giving an error number, the routine that the

error occurred in, and the mnemonic of the input parameter that is in error. The text associated with each error number is given in Table VI below. Among the types of input errors detected by the program are illegal mnemonics or class names, table sizes out of range, and missing variables on the Default File.

If any errors are detected, the Input Processor will attempt to read override data for any remaining cases to check the data for validity, and then it will abort the job. The user should protect himself by including the necessary job control cards both before and after an EXIT card if necessary to save any permanent files which may have been created prior to the case that had an error.

The following table defines the INPUTP error numbers and indicates the appropriate response to correct the error condition.

Table VI. INPUTP Error Number Definitions and Responses

ERROR NO.	DEFINITION AND USER RESPONSE
125	The input mnemonic printed was not found in the dictionary of legal mnemonics for the current keyword. The user should check to see if he misspelled the input parameter or included it under the wrong class name.
126	An unknown keyword was detected in the user input. The keyword listed was not defined as a class name on the Initialization File. The user may have misspelled a class name or placed an input mnemonic in column 1.
127	The table size (no. of rows) specified for the given table is greater than the maximum size defined on the Initialization File. The user should verify which size is correct and change his input or the Initialization File, as appropriate. (Remember that the sizes on the Initialization File should correspond to the actual dimension sizes in the program.)
128	An illegal number of table dimensions was read from the Initialization File for the given table. The number of dimensions read was not between 1 and 3, or the individual dimension sizes for a duplicate table name do not agree with those previously read. The user should correct the Initialization File, as appropriate.
129	An illegal START suffix was read from the Initialization File in connection with block data input. A character other than an S or T was suffixed to START. The user should correct his Initialization File to include a legal START statement.

ERROR NO.	DEFINITION AND USER RESPONSE
130	A default value was not read from the Default File for the given input variable. The given mnemonic was specified on the Initialization File but no default value was provided under its class name. The user may have misspelled the mnemonic on the Initialization File or on the Default File. He may also have placed the mnemonic in the wrong class name on one of the files.
131	An invalid "PARM" parameter was specified on the INPUTP execution card. Check for misspelling one of the legal parameters - TABLES, WARN, or LASTV.
132	A simple variable declared as octal on the Initialization File has not previously been assigned to a class name. The mnemonics of octal input variables must be mentioned under a class name before being mentioned under the OCTAL.SIMPLES keyword.
133	A table declared as octal on the Initialization File has not previously been defined. Each octal input table must be defined under the TABLES class name before its mnemonic is read under the OCTAL.TABLES keyword.
134	A simple variable declared as hexadecimal on the Initialization File has not previously been assigned to a class name. The mnemonics of hexadecimal input variables must be mentioned under a class name before being mentioned under the HEX.SIMPLES keyword.
135	A table declared as hexadecimal on the Initialization File has not previously been defined. Each hexadecimal input table must be defined under the TABLES class name before its mnemonic is read under the HEX.TABLES keyword.

REFERENCES

1. D. J. Lemoine, "Configuration Description Language Processor Design Disclosure", NSWC Technical Report 3881, Naval Surface Weapons Center, Dahlgren, Virginia, May 1979.

2. R. T. Bevan and J. H. Reynolds, "Computer Programming and Coding Standards for the FORTRAN and SIMSCRIPT II.5 Programming Languages", NSWC Technical Report 3878, Naval Surface Weapons Center, Dahlgren, Virginia. (To be Published)

3. "UPDATE Reference Manual", Control Data Corporation, No. 60342500, Sunnyvale, California, December 1975.

APPENDIX A

SAMPLE INITIALIZATION FILE

(SIMU11)

```
SHIP
    PHIO      SINS      LAMBDO
OUTPUT
    OUTLVL
    LVLSUP    FULREP
TARGET
    TPID      NFP
VECTORS
    RTSV      VTSV
TABLES
    C1DRAG    2    3    4
    BWNS      1    4
    SELOPT    1    8
    FPOPT     3    2    4    2
PRINT.OPTIONS
    MNEMONICS     LVLSUP     YES
    REPORT.TYPE    FULREP    YES
BLOCK.DATA
STARTS
         SUBROUTINE BLKS
/SHIPS
C                /SHIPS/ SHIP POSITION AND TYPE
         COMMON /SHIPS/ PHIO, LAMBDO, SINS
         REAL    LAMBDO
         INTEGER SINS
/TGT
C                /TGT/ TARGET PACKAGE INFORMATION
         COMMON /TGT/ TPID, NFF
         INTEGER TPID
/OPTIONS
C                /OPTIONS/ OUTPUT OPTIONS
         COMMON /OPTIONS/ OUTLVL, LVLSUP, FULREP
         INTEGER FULREP, OUTLVL
/MSLPOS
C                /MSLPOS/ MISSILE POSITION AND VELOCITY
         COMMON /MSLPOS/ RTSV(3), VTSV(3)
ENDS
STARTT
            BLOCK DATA BLKT
/OPTS
C                /OPTS/ INPUT OPTIONS
         COMMON /OPTS/ SELOPT(8), FPOPT(2,4,2)
         INTEGER SELOPT, FPOPT
/TBLES
C                /TBLES/ WIND AND DRAG TABLES
         COMMON /TBLES/ BWNS(4), C1DRAG(3,4)
ENDT
```

APPENDIX B

SAMPLE DEFAULT FILE

(SIMU13)

```
SHIP
    PHIO      0.5235                 SHIP LATITUDE
    LAMBDO    1.0                    SHIP LONGITUDE
    SINS      MK2                    NAVIGATION SYSTEM
TARGET
    TPID     900                     TARGET PACKAGE ID
    NFP      2                       NUMBER OF FOOTPRINTS
OUTPUT
    OUTLVL     2                     OUTPUT LEVEL
    LVLSUP     NO                    SUPPRESS MNEMONICS FLAG
    FULREP     NO                    FULL REPORT OPTION
VECTORS
    RTSV                             INITIAL MISSILE POSITION
          3.528E06      2.8356E05       -3.104E08
    VTSV                             INITIAL MISSILE VELOCITY
          1.37E04       3.320E02        -1.373E04
TABLES
    BWNS     4                       WIND TABLE
      5.0      6.5      4.3      -2.1
    SELOPT    8                      MISSILE SELECTION OPTION
       NORMAL     NORMAL     INSTANT     NORMAL
       INSTANT    NORMAL     NORMAL      NORMAL
    C1DRAG    3                      C1 DRAG TABLE
       .418     -.323     -.409     1.918
       .301      .783     -.549     1.585
       .117      .484     -.3078    1.548
    FPOPT    2                       FOOTPRINT OPTIONS
        31    30    51    41
        31    20    21    51
        51    50    41    41
        51    40    10    21
```

APPENDIX C

SAMPLE USER OVERRIDE INPUT

```
TITLE
   FIRST SAMPLE CASE
TARGET
   TPID    905
SHIP
   SINS    MK3      LAMBDC    .8752
TABLES
   BWNS    4
   4.2   6.5   4.3   -1.8
END

TABLES
   SELOPT   2
     INSTANT   INSTANT
VECTORS
   RTSV    4.01E07     3.01E07     -3.104E08
TITLE
   SECOND SAMPLE CASE
TARGET
   TPID    906
END
```

APPENDIX D

SAMPLE OUTPUT

```
                              + + + VARIABLES + + +

SHIP
    *LAMBDO   8.7528000000000E-01     PHIO     5.2350000000000E-01    *SINS    MK3
OUTPUT
    FULREP  NO                        LVLSUP  NO                      OUTLVL                2
TARGET
    NFP                         2    *TPID                     905

                              + + + VECTORS + + +

    RTSV    3.5280000000000E+06           2.8356000000000E+05           -3.1040000000000E+08
    VTSV    1.3700000000000E+04           3.3200000000000E+02           -1.3730000000000E+04

                              + + + TABLES + + +

                              ***   *BWNS    ***
            4.2000000000000E+00         6.5000000000000E+00         4.3000000000000E+00         -1.8000000000000E+00
                              ***   C1DRAG   ***
    ROW  1
            4.1800000000000E-01        -3.2300000000000E-01        -4.0900000000000E-01          1.9180000000000E+00
    ROW  2
            3.0100000000000E-01         7.8300000000000E-01        -5.4900000000000E-01          1.5850000000000E+00
    ROW  3
            1.1700000000000E-01         4.8400000000000E-01        -3.0780000000000E-01          1.5480000000000E+00
                              ***   FPOPT    ***
  BLOCK  ( I,  J,   1)
    ROW  1
                           31                          30                          51                          41
    ROW  2
                           31                          20                          21                          51
```

```
BLOCK ( I, J, 2)
   ROW  1
                    51                    50                    41                    41
   ROW  2
                    51                    40                    10                    21
                                   ***  SELOPT  ***
        NORMAL               NORMAL               INSTANT              NORMAL
        INSTANT              NORMAL               NORMAL               NCRMAL
```

```
SECOND SAMPLE CASE                                                    04/03/78  09.59.41.       PAGE  1

                                      + + + VARIABLES + + +

TARGET
  *TPID                  986

                                      + + + VECTORS + + +

  *RTSV    4.0100000000000E+07          3.0100000000000E+07          -3.1040000000000E+08

                                      + + + TABLES + + +

                                      ***  *SELOPT   ***
           INSTANT                INSTANT                INSTANT                NCRMAL
           INSTANT                NORMAL                 NORMAL                 NORMAL
```

APPENDIX E

SAMPLE OF GENERATED BLOCK DATA ROUTINES

```
      SUBROUTINE BLKS
C              /SHIPS/ SHIP POSITION AND TYPE
      COMMON /SHIPS/ PHIO, LAMBDO, SINS
      REAL    LAMBDO
      INTEGER SINS
C              /TGT/ TARGET PACKAGE INFORMATION
      COMMON /TGT/ TPID, NFP
      INTEGER TPID
C              /OPTIONS/ OUTPUT OPTIONS
      COMMON /OPTIONS/ OUTLVL, LVLSUP, FULREP
      INTEGER FULREP, OUTLVL
C              /MSLPOS/ MISSILE POSITION AND VELOCITY
      COMMON /MSLPOS/ RTSV(3), VTSV(3)
C     SHIP
      DATA LAMBDO /  1.000000000000E+00/
      DATA PHIO   /  5.235000000000E-01/
      DATA SINS   / 10HMK2          /

C     OUTPUT
      DATA FULREP / 10HNO           /
      DATA LVLSUP / 10HNO           /
      DATA OUTLVL /                 2   /

C     TARGET
      DATA NFP    /                 2   /
      DATA TPID   /               900   /

C     VECTORS
      DATA RTSV
     1 /      3.528000000000E+06, 2.835600000000E+05,
     2        -3.104000000000E+08/
      DATA VTSV
     1 /      1.370000000000E+04, 3.320000000000E+02,
     2        -1.373000000000E+04/
      END
```

```
      BLOCK DATA BLKT
C           /OPTS/ INPUT OPTIONS
      COMMON /OPTS/ SELOPT(8), FPOPT(2,4,2)
      INTEGER SELOPT, FPOPT
C           /TBLES/ WIND AND DRAG TABLES
      COMMON /TBLES/ BWNS(4), C1DRAG(3,4)

      DATA BWNS    (    1) / 5.0000000000000E+00/
      DATA BWNS    (    2) / 6.5000000000000E+00/
      DATA BWNS    (    3) / 4.3000000000000E+00/
      DATA BWNS    (    4) /-2.1000000000000E+00/

      DATA (C1DRAG (   1,J),J=1, 4) /
     1 4.1800000000000E-01,-3.2300000000000E-01,-4.0900000000000E-01,
     2 1.9180000000000E+00/
      DATA (C1DRAG (   2,J),J=1, 4) /
     1 3.0100000000000E-01, 7.8300000000000E-01,-5.4900000000000E-01,
     2 1.5850000000000E+00/
      DATA (C1DRAG (   3,J),J=1, 4) /
     1 1.1700000000000E-01, 4.8400000000000E-01,-3.0780000000000E-01,
     2 1.5480000000000E+00/

      DATA (FPOPT  (   1,J,   1),J=1, 4) /
     1          31,                30,                51,
     2          41/
      DATA (FPOPT  (   2,J,   1),J=1, 4) /
     1          31,                20,                21,
     2          51/
      DATA (FPOPT  (   1,J,   2),J=1, 4) /
     1          51,                50,                41,
     2          41/
      DATA (FPOPT  (   2,J,   2),J=1, 4) /
     1          51,                40,                10,
     2          21/

      DATA SELOPT (    1) /10HNORMAL     /
      DATA SELOPT (    2) /10HNORMAL     /
      DATA SELOPT (    3) /10HINSTANT    /
      DATA SELOPT (    4) /10HNORMAL     /
      DATA SELOPT (    5) /10HINSTANT    /
      DATA SELOPT (    6) /10HNORMAL     /
      DATA SELOPT (    7) /10HNORMAL     /
      DATA SELOPT (    8) /10HNORMAL     /
      END
```

APPENDIX F

SAMPLE BEGIN/REVERT PROCEDURE FOR EXECUTION

```
LINE
 NO.
----

  1    MODELXEQ,*I=INPUT,*L=OUTPUT,*TABLES=NO,*WARN=NO,
       *LASTV=YES,*XEQ=YES.
  2    BEGIN,INIT,*FILE.
  3    BEGIN,EXEC,*FILE,I=*I,L=*L,T=*TABLES,WARN=*WARN,
                   LASTV=*LASTV,XEQ=*XEQ.
  4    REVERT.
  5    EXIT,S.
  6    REVERT,ABORT.
  7    7/8/9

  8    INIT.
  9    RETURN,INPUTP,MODEL.
 10    ATTACH,INPUTP,ID=NTH.
 11    ATTACH,MODEL,COMPUTATIONAL PROGRAM.
 12    RETURN,SIMU11,SIMU13,SIMU19,SIMU17.
 13    RETURN,SIMU3,SIMU2,DEFDAT.
 14    ATTACH,SIMU11,MODEL INITIALIZATION FILE.
 15    ATTACH,SIMU13,MODEL DEFAULT FILE.
 16    ATTACH,DEFDAT,RELOCATABLE BINARY OF DEFAULT BLOCK DATA.
       /* ATTACH ANY DATA FILES NEEDED BY MODEL
 17    REVERT.
 18    EXIT,S.
 19    REVERT,ABORT.
 20    7/8/9

 21    EXEC,*I=INPUT,*L=OUTPUT,*T=NO,*WARN=NO,*LASTV=YES,
       XEQ=YES.
 22    RETURN,SIMU21,SIMU23,ZZZZ.
 23    REWIND,LGO21,LGO23.
 24    INPUTP,SIMU5=*I,SIMU6=*L,PARM,TABLES=*T,LASTV=*LASTV,WARN=*WARN.
 25    IFC(EQ,*XEQ,YES,TEST)
 26       IF(FILE,SIMU21,GENS)
 27          REWIND,SIMU21.
 28          FTN,I=SIMU21,A,L=ZZZZ,B=LGO21.
 29       ENDIF(GENS)
 30       IF(FILE,SIMU23,GENT)
 31          REWIND,SIMU23.
 32          FTN,I=SIMU23,A,L=ZZZZ,B=LGO23.
 33       ENDIF(GENT)
 34       LOAD(DEFDAT)
 35       LOAD(LGO21)
 36       LOAD(LGO23)
 37       MODEL.
 38    ENDIF(TEST)
       /* EXECUTE ANOTHER CASE IF NECESSARY
 39    IF(-FILE,SIMU2,NXTCAS)
```

```
      BEGIN,EXEC,*FILE,I=*I,L=*L,T=*T,WARN=*WARN,LASTV=*LASTV,
      XEQ=*XEQ.
ENDIF(NXTCAS)
REVERT.
EXIT,S.
REVERT,ABORT.
7/8/9
```

DISTRIBUTION

DEFENSE DOCUMENTATION CENTER
CAMERON STATION
ALEXANDRIA, VIRGINIA 22314 (12)

DEFENSE PRINTING SERVICE
WASHINGTON NAVY YARD
WASHINGTON, DC 20374

LIBRARY OF CONGRESS
WASHINGTON, DC 20540
ATTN: GIFT AND EXCHANGE DIVISION (4)

NAVAL AIR DEVELOPMENT CENTER
WARMINSTER, PENNSYLVANIA 18974
ATTN: TECHNICAL LIBRARY

NAVAL OCEAN SYSTEMS CENTER
271 CATALINA BOULEVARD
SAN DIEGO, CALIFORNIA 92152
ATTN: CODE 91

NAVAL SHIP RESEARCH & DEVELOPMENT CENTER
BETHESDA, MARYLAND 20084
ATTN: TECHNICAL LIBRARY

NAVAL TRAINING EQUIPMENT CENTER
ORLANDO, FLORIDA 32813
ATTN: TECHNICAL LIBRARY (CODE N-74)

CHARLES STARK DRAPER LABORATORY
68 ALBANY STREET
CAMBRIDGE, MASSACHUSETTS 02139
ATTN: JANE GOODE
DONALD MILLARD
WILLIAM OSTANEK
THEODORE PETRANIC
THOMAS THORVALDSEN
PETER VOLANTE

EG&G, INCORPORATED
WASHINGTON ANALYTICAL SERVICES CENTER
P.O. BOX 552
DAHLGREN, VIRGINIA 22448
ATTN: FRANK EAK
TIMOTHY GILBERT
DAVID LAWRENCE
ASHTON RUDD
STEVEN SCANZONI

GENERAL ELECTRIC/ORDNANCE SYSTEMS
ELECTRONIC SYSTEMS DIVISION
100 PLASTICS AVENUE
PITTSFIELD, MASSACHUSETTS 01201
ATTN: PAUL S. SCHUBERT
PHILIP ARNDT

DISTRIBUTION

SPERRY UNIVAC, INCORPORATED
DAHLGREN, VIRGINIA 22448
ATTN: DALE HARRIS
STANLEY ROGOFF

FCDSSA, DAM NECK
VIRGINIA BEACH, VIRGINIA 23461
ATTN: MR. H. J. STANSELL (CODE 00T)

LOCAL:

E41

F10
F18

K
K02
K10
K20
K30
K31
K32
K50
K51
K53
K54
K55
K56
K60
K64
K70
K71 (20)
K72 (15)
K73 (5)
K74

N20
N22

X210 (2)